ARL-TN-0913 • Sep 2018

US Army Research Laboratory

AppPottsRS: A Read–Shockley Class for SPPARKS

by Efraín Hernández–Rivera

NOTICES

Disclaimers

ARL-TN-0913 ● SEP 2018

US Army Research Laboratory

AppPottsRS: A Read–Shockley Class for SPPARKS

by Efraín Hernández–Rivera
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE	***Form Approved*** ***OMB No. 0704-0188***

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED (From - To)**
September 2018	Technical Note	January 2017–January 2018

4. TITLE AND SUBTITLE
AppPottsRS: A Read–Shockley Class for SPPARKS

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Efraín Hernández–Rivera

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-WMM-B
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TN-0913

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
primary author's email: <efrain.hernandez18.civ@mail.mil>.

14. ABSTRACT
The Stochastic Parallel PARticle Kinetic Simulator (SPPARKS) framework has enabled large-scale simulation of microstructural evolution as simulated by the Potts Monte Carlo model. In order to more accurately model microstructural evolution, the AppPottsRS code was developed as an extension to the SPPARKS' Potts model (AppPotts). The AppPottsRS class uses a more robust microstructural description based on Euler–Bunge angles and relies on the Read–Shockley equation to characterize interfacial energies. This technical note briefly describes the AppPottsRS class, which is used to model grain growth under the SPPARKS framework. This includes a detailed description of an example input file used to simulate grain growth of a hexagonal material. Based on ongoing efforts, the code was adapted to couple texture interactions to magnetic fields. A short example is provided where magnetic-field-enhanced grain growth is shown. Lastly, the C++ source code is included as an appendix.

15. SUBJECT TERMS
Potts Monte Carlo, Read–Shockley, C++, grain growth, magnetic field

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Efraín Hernández–Rivera
Unclassified	Unclassified	Unclassified	UU	36	**19b. TELEPHONE NUMBER (Include area code)** 410-306-4961

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

Acknowledgments

The author would like to thank Drs Philip Goins (Oak Ridge Associated Universities/US Army Research Laboratory) and Jeff Allen (US Army Engineer Research and Development Center) for useful discussions in the development of the code presented here and/or previous versions.

1. Introduction

Understanding how microstructures evolve is essential for designing better performing materials, as these are known to influence material properties. Multiple computational codes have been developed to model how materials evolve, to include Sandia National Laboratories' Stochastic Parallel PARticle Kinetic Simulator (SPPARKS) framework.[1] This framework simulates microstructural evolution using the well-known Potts Monte Carlo (PMC) model.[2] While a highly efficient and scalable algorithm, SPPARKS' PMC model implements an isotropic interfacial energy Hamiltonian. This simplified implementation has been successfully used to model a wide range of processing conditions (e.g., welding).[3] Nonetheless, a more robust implementation that enables texture-specific interfacial energies would enable higher-fidelity simulations. This is routinely done by defining grain boundary misorientation angles and using the Read–Shockley equation to determine the grain boundary energies.[4–7] A Read–Shockley-based PMC algorithm was developed to model microstructural evolution under the SPPARKS framework, termed the *AppPottsRS*.

This technical note documents how the Read–Shockley PMC algorithm was implemented into SPPARKS. A short description of a sample input file that simulates evolution of a hexagonal material is provided for clarity. The code was extended to include how material textures "couple" to external fields (e.g., magnetic fields), and a brief example is provided. The source `C++` code is included as an appendix. Note that the AppPottsRS class has been developed to run under the SPPARKS framework (i.e., it must be compiled under SPPARKS). Lastly, it should be noted that a separate technical report[8] documents AppPottsRS' validation and usage to model microstructural evolution under thermal gradients.

2. Algorithm Implementation

As previously mentioned, this code was written to run as a derived-class under the SPPARKS framework. In fact, it was built similar to the AppPottsNeighOnly class and is a child class to AppPotts. Therefore, AppPottsRS must be included into SPPARKS' source directory and compiled using the appropriate `makefile`, as it relies on the framework to build the computational domain, define global variables, and handle the communication. While the Read–Shockley PMC model has been extensively studied, to the best of the author's knowledge, this is the first publicly available implementation into SPPARKS. Following is a brief description of the input file, shown in Fig. 1.

As this class was derived from AppPotts (standard PMC), there are several similarities between the input file shown here and the one in SPPARKS' standard distribution. For simplicity, the following list describes the lines of interest as numbered in Fig. 1.

4: seed value for random number generator

6: logfile name for saving generic run information

13: call to class (potts/rs), assigning number of possible orientations (one per voxel), and define material symmetry (hexagonal)

15: computational domain dimension

16: define lattice connectivity/neighborhood and grid spacing

17: define rectangular domain size

18–19: create domain

24: assign a unique grain orientation flag per voxel

26–28: assign uniformly random distribution for Euler angles, which are redistributed to generate a random texture, yielding the distributions as shown in Fig. 2.

31–32: flags for sampling sites

36: cutoff angle for Read–Shockley equation

39–40: mobility parameters, as defined by Humphreys[9]

42: scaling coefficient for the Potts' Hamiltonian

45: call to diagnostic calculation, which outputs global energy to logfile

48: define computational temperature ($k_B T$)

51–54: output specific commands

56: computational run time

```
# SPPARKS AppPottsRS tests on hexagonal symmetry

#Seed for RN generator
seed                56789

log                 example.log

#nspins is same as in parent class
#
#symm is a flag to assign symmetry
## cubic = 24 (default) | hexagonal = 12
#                              nspins      symm
app_style           potts/rs   2097152     12

dimension           3
lattice             sc/26n 1.0
region              box block 0 128 0 128 0 128

create_box          box
create_sites        box

#spin: grain orientation ID
#each spin is related to a set of Euler angles
set                 site unique
#Euler angles: d1 = ph1, d2 = Phi and d3 = phi2
set                 d1   range 0 1
set                 d2   range 0 1
set                 d3   range 0 1

#kMC sampling (sweeping) algorithm
sweep               random
sector              yes

#app commands
#cutoff for high/low angle GBs
cutoff              20.0
#Humpherys mobility
#Mobility = M0 * [1 - exp(-scale * pow(thetar,expo))]
mobility            expo 4
mobility            scale 5
#Potts Hamiltonian scaling factor J = n * sum(1-delta_ij)
energy_scaling      1

#Diagnostics
diag_style          energy

#PMC temperature, i.e. for Boltzmann statistics
temperature         0.25

#Output commands
stats               0.00001

dump                1 text 1.0 potts_rs.*.dump id i1 d1 d2 d3 energy
diag_style          cluster delt 1.0 stats no logfreq 9 10.0 filename cluster.dat

run                 100
```

Fig. 1 Example input file used to run AppPottsRS, which generates a hexagonal symmetry (material) meshless $128 \times 128 \times 128$ domain and runs for 2600 MCS ($t_{\mathbf{comp}} = 100$)

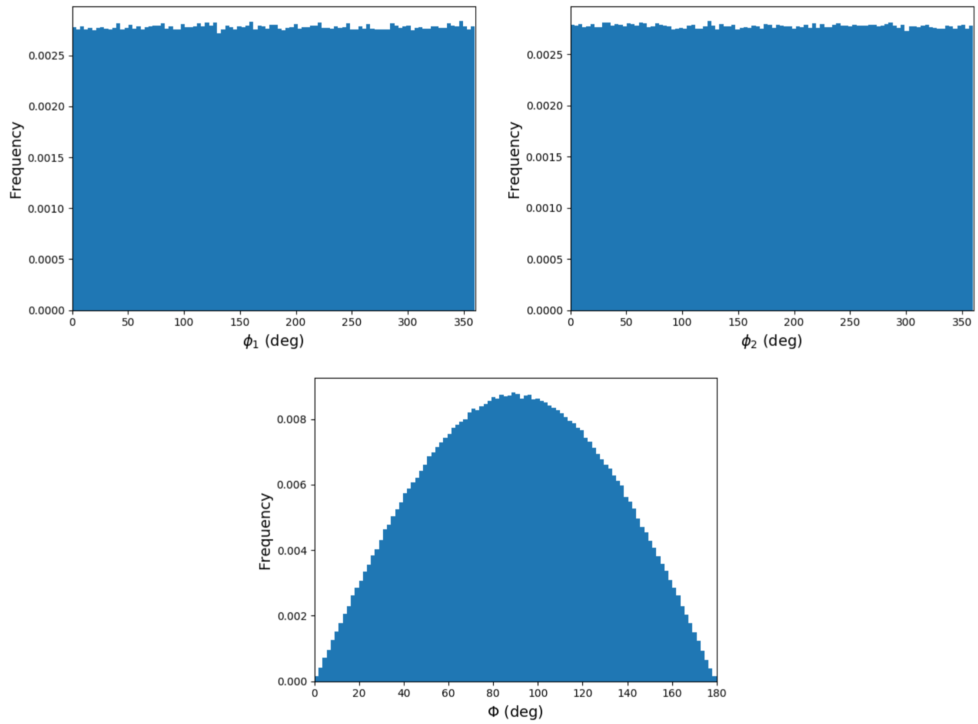

Fig. 2 Normalized distributions of the Euler angles for a randomly textured material

Figure 3 shows a representative grain growth process of a $128 \times 128 \times 128$ domain that was executed for run $= t_{\text{comp}} = 100$. The microstructures have been thresholded in such a way that only voxel with low energies are shown (i.e., filtering out grain boundaries). Since each voxel has a unique orientation at the beginning of the simulation, this simulation could be thought of as solidification of a material. At early times, many small grains are growing giving the appearance of small precipitate-like clusters. Towards the end of the simulation, clearly distinguished grains can be identified. For a complete description of grain growth simulations using AppPottsRS, the reader is directed to the technical report by Hernández–Rivera et al.[8]

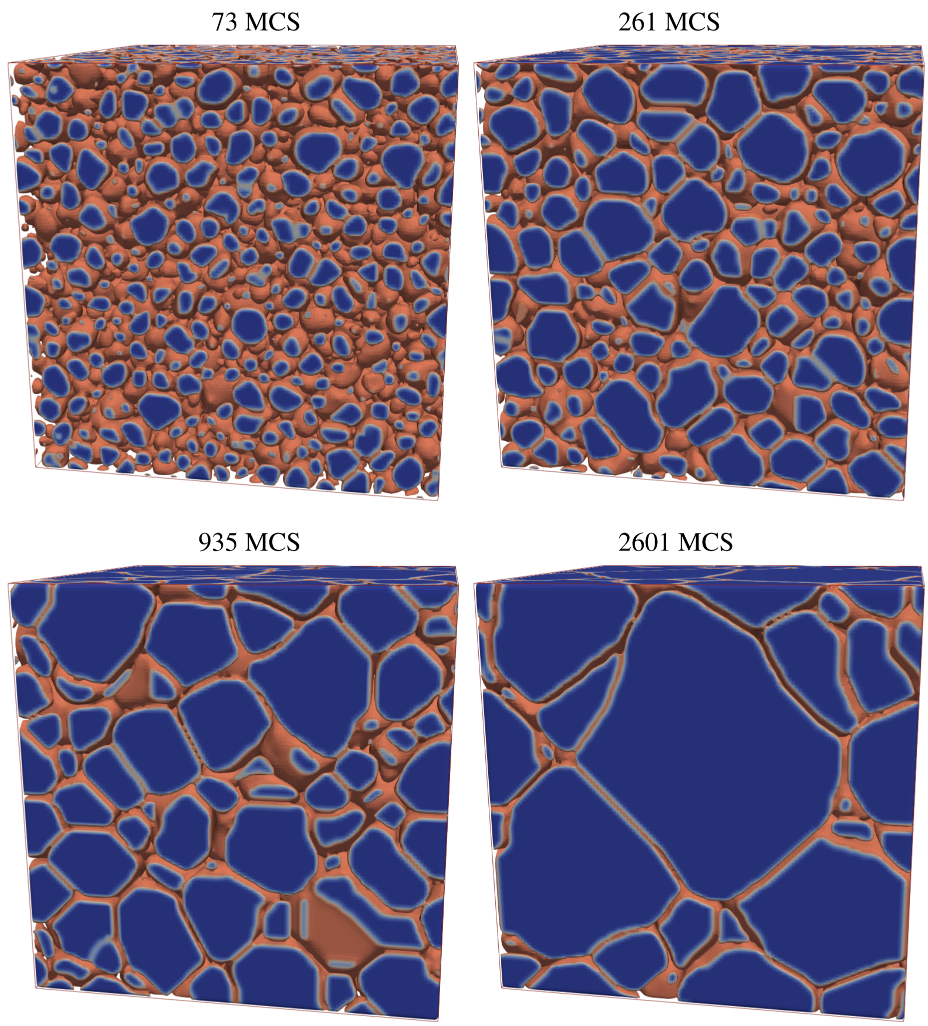

Fig. 3 Representative simulated microstructural evolution showing low-energy voxels (i.e., isolating grains)

Another example application of this code is modeling under different processing conditions (e.g., grain growth under magnetic fields). Similar to the work by Lei et al.,[10] the texture description of the microstructure was coupled to the direction of an applied magnetic field. Figure 4 shows the grain growth curves for the case of a slightly positive ($+\mathbf{H}$), negative ($-\mathbf{H}$), and zero ($\mathbf{H} = 0$) applied magnetic fields aligned the material's normal direction. While Lei reports that magnetic fields have no influence on the grain growth kinetics, Fig. 4 shows that the magnetic field will increase the kinetics. This is in agreement with Goins et al.[11] The Lei findings that kinetics are uninfluenced by magnetic fields are likely due to the limited run time where kinetics appear similar (i.e., $t_{\text{run}} < 250$ MCS).

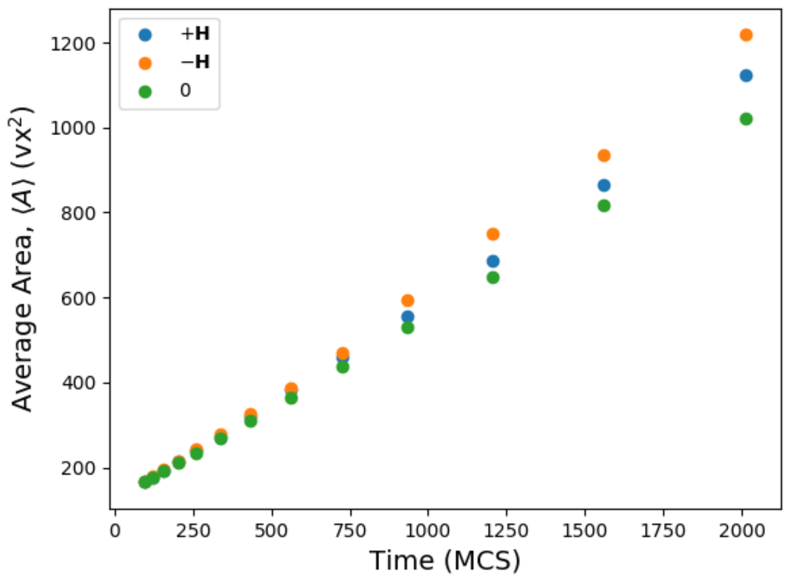

Fig. 4 Grain growth curve for different field strengths

3. Summary

A more accurate description of grain boundaries for microstructural modeling can be obtained through use of the Read–Shockley equation. This technical note outlines the development of AppPottsRS, which incorporates the Read–Shockley equation into the PMC model currently available in SPPARKS. A representative microstructure is shown and briefly described. The use of a more robust texture-based description enables coupling to magnetic fields, which are shown to accelerate the growth kinetics. Finally, the source code is provided as an appendix.

4. References

1. Sandia National Laboratories. SPPARKS documentation. [accessed 2018 Jul 26]. http://spparks.sandia.gov/.

2. Potts RB. Some generalized order-disorder transformations. Math Proc Cambridge Phil Soc. 1952;48:106–109.

3. Sandia National Laboratories. SPPARKS documentation: app_style potts_weld. [accessed 2018 Jul 26]. https://spparks.sandia.gov/doc/app_potts_weld.html.

4. Anderson M, Srolovitz D, Grest G, Sahni P. Computer simulation of grain growth-I. Kinetics. Acta Metallurgica. 1984;32(5):783–791.

5. Srolovitz D, Anderson MP, Sahni PS, Grest GS. Computer simulation of grain growth-II. Grain size distribution, topology, and local dynamics. Acta Metallurgica. 1984;32(5):793–802.

6. Grest G, Srolovitz D, Anderson M. Computer simulation of grain growth-IV. Anisotropic grain boundary energies. Acta Metallurgica. 1985;33(3):509–520.

7. Holm EA, Hassold GN, Miodownik MA. On misorientation distribution evolution during anisotropic grain growth. Acta Materialia. 2001;49(15):2981–2991.

8. Hernández-Rivera E, Goins PE, Murdoch HA. Anisotropic grain growth modeling under the SPPARKS framework. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2018 Sep. Report No.: ARL-TR-8507.

9. Humphreys F. A unified theory of recovery, recrystallization and grain growth, based on the stability and growth of cellular microstructures–I. The basic model. Acta Materialia. 1997;45(10):4231–4240.

10. Lei H, Zhu X, Sun Y, Hu L, Song W. Effects of magnetic field on grain growth of non-ferromagnetic metals: A Monte Carlo simulation. Europhys Lett. 2009;85(3):38004.

11. Goins PE, Murdoch HA, Hernández-Rivera E, Tschopp MA. Effect of magnetic fields on microstructure evolution. Comp Mat Sci. 2018;150:464–474.

Appendix A. AppPottsRS Class Source File

```
/* ---------------------------------------------------------
 AppPottsRS class source - a SPPARKS Read--Shockley
                           implementation
 Developed by Efrain Hernandez-Rivera (2017--2018)
 US Army Research Laboratory
 --
 THIS SOFTWARE IS MADE AVAILABLE ON AN "AS IS" BASIS
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, NEITHER
 EXPRESSED OR IMPLIED
------------------------------------------------------- */

#include "stdio.h"
#include "string.h"
#include "stdlib.h"
#include "domain.h"
#include "math.h"
#include "app_potts_rs.h"
#include "random_park.h"
#include "comm_lattice.h"
#include "error.h"

using namespace SPPARKS_NS;

#define MY_PI  3.14159265358979323846 // pi
#define MY_2PI 6.28318530717958647692 // 2pi

/* ---------------------------------------------------- */

AppPottsRS::AppPottsRS(SPPARKS *spk, int narg, char **arg) :
AppPotts(spk,narg,arg)
{
  ninteger = 1;
  //1 double array per Euler angle
  ndouble = 3;
```

```
  // add the extra arrays
  recreate_arrays();

  // only error check for this class, not derived classes
  if (strcmp(arg[0],"potts/rs") == 0 && narg < 2)
  error->all(FLERR,"Illegal␣app␣style␣command");

  //cutoff misorientation angle
  thetam=15.0/180.0*MY_PI;

  //interaction (interfacial) energy
  Jij=1.0;

  //Mobility parameters
  nmob=4.0;   bmob=5.0;

  //Symmetry operator
  Osym=24;
  if (narg == 3)
  Osym=atoi(arg[2]);
}

/* ----------------------------------------------------------
 Destructor
---------------------------------------------------------- */
AppPottsRS::~AppPottsRS()
{
  //free up memory from quaternion symmetry operator
  for (int i = 0; i<Osym; i++)
    delete[] symquat[i];

  delete[] symquat;
}

/* ----------------------------------------------------------
```

```
 Initialize before each run
 check validity of site values
------------------------------------------------------- */

void AppPottsRS::init_app()
{
  delete [] sites;
  delete [] unique;
  sites = new int[1 + maxneigh];
  unique = new int[1 + maxneigh];

  dt_sweep = 1.0/maxneigh;

  int flag = 0;

  //Check angles are within corresponding range
  //originally should be set to phi=U(0,1)
  for (int i = 0; i < nlocal; i++) {
    //Randonly distribute Euler angles
    phi1[i]=MY_2PI*phi1[i]; Phi[i]=acos(2*Phi[i]-1);
    phi2[i]=MY_2PI*phi2[i];
    if (phi1[i] < 0 || phi1[i] >= MY_2PI) flag = 1;
    if (phi2[i] < 0 || phi2[i] >= MY_2PI) flag = 1;
    if (Phi[i]  < 0 || Phi[i]  >= MY_PI)  flag = 1;
  }

  //Initialize symmetry operator as quaternion vectors
  //Osym = 24 (cubic), 12 (hexagonal)
  symmat(&symquat);

  comm->all();

  int spi, spn, nei;
  double qi[4], qj[4];

  for (int i=0; i<nlocal+nghost; i++) {
```

```
    spi=spin[i];
    euler2quat(i,  qi);

    for (int n=0; n<numneigh[i]; n++) {
      nei=neighbor[i][n];
      spn=spin[nei];

      //order by min/max to avoid duplicate pairs
      int smin = MIN(spi,spn);
      int smax = MAX(spi,spn);

      std::pair <int, int> spins = std::make_pair(smin,smax);

      if (spi != spn && misos.count(spins) == 0) {
        euler2quat(nei,qj);

        //insert misorientation angle
        //between spin ID pair (thetar)
        misos[spins]=quaternions(qi,qj)/thetam;
      }
    }
  }

  if (logfile)
    fprintf(logfile,"␣␣Pairs␣misorientation␣map␣created\n");
  if (screen && me==0)
    fprintf(screen,"␣␣Pairs␣misorientation␣map␣created\n");

  int flagall;
  MPI_Allreduce(&flag,&flagall,1,MPI_INT,MPI_SUM,world);
  if (flagall)
    error->all(FLERR,"One␣or␣more␣sites␣have␣invalid␣values");
}

/* ----------------------------------------------------------
```

```
 Set site value ptrs each time iarray/darray are
 reallocated
------------------------------------------------------- */
void AppPottsRS::grow_app()
{
  // set pointers
  // to define these, use command
  //  create_sites box iN and set iN
  spin = iarray[0];
  phi1 = darray[0];
  Phi  = darray[1];
  phi2 = darray[2];
}

/* -------------------------------------------------------
 User defined optional parameters
------------------------------------------------------- */
void AppPottsRS::input_app(char *command, int narg, char **arg)
{
  if (narg < 1) {
    error->all(FLERR,"Invalid␣command␣for␣app_style");
  }

  //Redefine mobility parameters (n,b)
  if (strcmp(command,"mobility") == 0) {
    if (narg != 2)
      error->all(FLERR,"Illegal␣mobility␣flag:␣requires␣"
        "two␣arguments,␣parameter-flag␣and␣parameter-value,␣"
        "(e.g.␣mobility␣expo␣3.0)\n");

    if (strcmp(arg[0],"expo") == 0) {
      nmob=atof(arg[1]);

      if (logfile)
        fprintf(logfile,"␣␣Mobility␣exponent␣reset␣to␣%g\n",nmob);
```

```
      if (screen && me==0)
        fprintf(screen,"␣␣Mobility␣exponent␣reset␣to␣%g\n",nmob);

    }
    else if (strcmp(arg[0],"scale") == 0) {
      bmob=atof(arg[1]);

      if (logfile)
        fprintf(logfile,"␣␣Mobility␣scaling␣reset␣to␣%g\n",bmob);
      if (screen && me==0)
        fprintf(screen,"␣␣Mobility␣scaling␣reset␣to␣%g\n",bmob);
    }
    else
      error->all(FLERR,"Mobility␣parameter␣not␣recognized\n");
  }
  //Cutoff angle for Read-Shockley
  else if (strcmp(command,"cutoff") == 0) {
    if (narg<1)
      error->all(FLERR,"Illegal␣cutoff␣angle␣command\n");
    thetam=fabs(atof(arg[0]))/180.0*MY_PI;
    if (thetam>MY_2PI)
      error->all(FLERR,"Cutoff␣angle␣must␣be␣defined␣in␣"
                   "terms␣of␣degrees␣(0,360)\n");

    if (logfile)
      fprintf(logfile,"␣␣Low-to-high␣angle␣cutoff␣reset␣"
                "to␣%s␣deg\n",arg[0]);
    if (screen && me==0)
      fprintf(screen,"␣␣Low-to-high␣angle␣cutoff␣reset␣"
                "to␣%s␣deg\n",arg[0]);
  }
  //Potts interfacial energy scaler
  else if (strcmp(command,"energy_scaling") == 0) {
    if (narg<1)
      error->all(FLERR,"Illegal␣scaling␣energy␣command\n");
```

```
    Jij=atof(arg[0]);
    if (Jij<0)
      error->all(FLERR,"Illegal␣energy␣value␣(>0)\n");

    if (logfile)
      fprintf(logfile,"␣␣PMC␣energy␣scaling␣by␣%g.\n",Jij);
    if (screen && me==0)
      fprintf(screen,"␣␣PMC␣energy␣scaling␣by␣%g.\n",Jij);
  }
  else
    error->all(FLERR,"Input␣command␣not␣recognized␣by␣app\n");
}

/* ----------------------------------------------------------
 Compute Hamiltonian of site
---------------------------------------------------------- */
double AppPottsRS::site_energy(int i)
{
  int nei;
  double eng = 0.0, qi[4], qj[4], thetar;

  euler2quat(i,qi);

  for (int j = 0; j < numneigh[i]; j++) {
    nei=neighbor[i][j];
    if (spin[i] == spin[nei]) continue;

    int smin = MIN(spin[i],spin[nei]);
    int smax = MAX(spin[i],spin[nei]);

    std::pair <int, int> spins = std::make_pair(smin,smax);

    // ratio of theta/theta_m
    if (misos.count(spins) == 1)
      thetar=misos[spins];
```

```
    else {
      euler2quat(nei,qj);
      thetar=quaternions(qi,qj)/thetam;
      misos[spins]=thetar;
    }

    if (thetar >= 1.0 || thetam<1e-8)
      eng+=1;
    else if (thetar > 0.0)
      eng+=thetar*(1.0-log(thetar));
  }

  return Jij*eng;
}

/* -------------------------------------------------------
 Convert symmetry matrix to quaternion form
------------------------------------------------------- */
void AppPottsRS::mat2quat(const double O[3][3], double q[4])
{
  double q4 = 0;
  if( (1 + O[0][0] + O[1][1] + O[2][2]) > 0) {
    q4   = sqrt(1 + O[0][0] + O[1][1] + O[2][2])/2;
    q[0] = q4;
    q[1] = (O[2][1] - O[1][2])/(4*q4);
    q[2] = (O[0][2] - O[2][0])/(4*q4);
    q[3] = (O[1][0] - O[0][1])/(4*q4);
  }
  else if ( (1 + O[0][0] - O[1][1] - O[2][2]) > 0) {
    q4   = sqrt(1 + O[0][0] - O[1][1] - O[2][2])/2;
    q[0] = (O[2][1] - O[1][2])/(4*q4);
    q[1] = q4;
    q[2] = (O[1][0] + O[0][1])/(4*q4);
    q[3] = (O[0][2] + O[2][0])/(4*q4);
  }
```

```
  else if ( (1 - O[0][0] + O[1][1] - O[2][2]) > 0) {
    q4   = sqrt(1 - O[0][0] + O[1][1] - O[2][2])/2;
    q[0] = (O[0][2] - O[2][0])/(4*q4);
    q[1] = (O[1][0] + O[0][1])/(4*q4);
    q[2] = q4;
    q[3] = (O[2][1] + O[1][2])/(4*q4);
  }
  else if ( (1 - O[0][0] - O[1][1] + O[2][2]) > 0) {
    q4 = sqrt(1 - O[0][0] - O[1][1] + O[2][2])/2;
    q[0] = (O[1][0] - O[0][1])/(4*q4);
    q[1] = (O[0][2] + O[2][0])/(4*q4);
    q[2] = (O[2][1] + O[1][2])/(4*q4);
    q[3] = q4;
  }
}

/* -------------------------------------------------------
 Define the symmetry operator
------------------------------------------------------- */
void AppPottsRS::symmat(double ***sym)
{
  //grow by number of symmetric operators
  (*sym) = new double*[Osym];

  //grow for symmetry quaternion vectors
  for (int o=0; o<Osym; o++)
    (*sym)[o] = new double[4];

  //buffer for quaternion
  double q[4];

  if (Osym == 24) {
    //cubic symmetry
    double SYM[24][3][3] =
      { {{ 1, 0, 0}, { 0, 1, 0}, { 0, 0, 1}},
```

```
        {{ 1, 0, 0}, { 0,-1, 0}, { 0, 0,-1}},
        {{ 1, 0, 0}, { 0, 0,-1}, { 0, 1, 0}},
        {{ 1, 0, 0}, { 0, 0, 1}, { 0,-1, 0}},
        {{-1, 0, 0}, { 0, 1, 0}, { 0, 0,-1}},
        {{-1, 0, 0}, { 0,-1, 0}, { 0, 0, 1}},
        {{-1, 0, 0}, { 0, 0,-1}, { 0,-1, 0}},
        {{-1, 0, 0}, { 0, 0, 1}, { 0, 1, 0}},
        {{ 0, 1, 0}, {-1, 0, 0}, { 0, 0, 1}},
        {{ 0, 1, 0}, { 0, 0,-1}, {-1, 0, 0}},
        {{ 0, 1, 0}, { 1, 0, 0}, { 0, 0,-1}},
        {{ 0, 1, 0}, { 0, 0, 1}, { 1, 0, 0}},
        {{ 0,-1, 0}, { 1, 0, 0}, { 0, 0, 1}},
        {{ 0,-1, 0}, { 0, 0,-1}, { 1, 0, 0}},
        {{ 0,-1, 0}, {-1, 0, 0}, { 0, 0,-1}},
        {{ 0,-1, 0}, { 0, 0, 1}, {-1, 0, 0}},
        {{ 0, 0, 1}, { 0, 1, 0}, {-1, 0, 0}},
        {{ 0, 0, 1}, { 1, 0, 0}, { 0, 1, 0}},
        {{ 0, 0, 1}, { 0,-1, 0}, { 1, 0, 0}},
        {{ 0, 0, 1}, {-1, 0, 0}, { 0,-1, 0}},
        {{ 0, 0,-1}, { 0, 1, 0}, { 1, 0, 0}},
        {{ 0, 0,-1}, {-1, 0, 0}, { 0, 1, 0}},
        {{ 0, 0,-1}, { 0,-1, 0}, {-1, 0, 0}},
        {{ 0, 0,-1}, { 1, 0, 0}, { 0,-1, 0}} };

    //initialize global operator
    for (int o=0; o<Osym; o++) {
      mat2quat(SYM[o],q);
      for (int i=0; i<4; i++)
        (*sym)[o][i]=q[i];
    }
  }
  else if (Osym == 12) {
    //hexagonal symmetry
    double a = sqrt(3)/2;
    double SYM[12][3][3] =
```

```
  { {{   1,    0,    0}, {    0,    1,    0}, {    0,    0,    1}},
  {{-0.5,    a,    0}, {   -a,-0.5,    0}, {    0,    0,    1}},
  {{-0.5,   -a,    0}, {    a,-0.5,    0}, {    0,    0,    1}},
  {{ 0.5,    a,    0}, {   -a, 0.5,    0}, {    0,    0,    1}},
  {{  -1,    0,    0}, {    0,   -1,    0}, {    0,    0,    1}},
  {{ 0.5,   -a,    0}, {    a, 0.5,    0}, {    0,    0,    1}},
  {{-0.5,   -a,    0}, {   -a, 0.5,    0}, {    0,    0,   -1}},
  {{   1,    0,    0}, {    0,   -1,    0}, {    0,    0,   -1}},
  {{-0.5,    a,    0}, {    a, 0.5,    0}, {    0,    0,   -1}},
  {{ 0.5,    a,    0}, {    a,-0.5,    0}, {    0,    0,   -1}},
  {{  -1,    0,    0}, {    0,    1,    0}, {    0,    0,   -1}},
  {{ 0.5,   -a,    0}, {   -a,-0.5,    0}, {    0,    0,   -1}} };

    //initialize global operator
    for (int o=0; o<Osym; o++) {
      mat2quat(SYM[o],q);
      for (int i=0; i<4; i++)
        (*sym)[o][i]=q[i];
    }
  }
}

double AppPottsRS::quaternions(const double qi[4], const double qj[4])
{
  double miso0, misom=MY_2PI;

  double q[4], qib[4], qjb[4], qmin[4]={0,0,0,0};
  for (int o1=0; o1<Osym; o1++) {
    for (int o2=0; o2<Osym; o2++) {
      quat_mult(symquat[o1],qi,qib);
      quat_mult(symquat[o2],qj,qjb);

      //j-grain conjugate quaternion
      qjb[1]=-qjb[1]; qjb[2]=-qjb[2]; qjb[3]=-qjb[3];
```

```
      quat_mult(qib,qjb,q);
      miso0 = 2*acos(q[0]);

      if (miso0 > MY_PI)
        miso0 = miso0-MY_2PI;
      if (fabs(miso0) < misom) {
        misom=fabs(miso0);
        qmin[0]=q[0]; qmin[1]=q[1]; qmin[2]=q[2]; qmin[3]=q[3];
      }

    }
  }

  miso0=2*acos(qmin[0]);
  if (miso0 > MY_PI)
    miso0=miso0-MY_2PI;

  return fabs(miso0);
}

void AppPottsRS::quat_mult(const double qi[4], const double qj[4],
                           double q[4])
{
  //Hamilton multiplication/product
  //multiplying quaternions and update
  q[0] = qi[0]*qj[0] - qi[1]*qj[1] - qi[2]*qj[2] - qi[3]*qj[3];
  q[1] = qi[0]*qj[1] + qi[1]*qj[0] + qi[2]*qj[3] - qi[3]*qj[2];
  q[2] = qi[0]*qj[2] - qi[1]*qj[3] + qi[2]*qj[0] + qi[3]*qj[1];
  q[3] = qi[0]*qj[3] + qi[1]*qj[2] - qi[2]*qj[1] + qi[3]*qj[0];
}

void AppPottsRS::euler2quat(int i, double q[4])
{
  //Convert grain Euler angles to quaternion vector
  double p1=phi1[i], P=Phi[i], p2=phi2[i];
```

```
  q[0]=cos(P/2.)*cos((p1+p2)/2.);
  q[1]=sin(P/2.)*cos((p1-p2)/2.);
  q[2]=sin(P/2.)*sin((p1-p2)/2.);
  q[3]=cos(P/2.)*sin((p1+p2)/2.);
}

/* -------------------------------------------------------
 rKMC method
 perform a site event with no null bin rejection
 flip to random neighbor spin without null bin
------------------------------------------------------- */
void AppPottsRS::site_event_rejection(int i, RandomPark *random)
{
  int    oldstate=spin[i];
  double iphi[3]={phi1[i],Phi[i],phi2[i]};

  // events = spin flips to neighboring site different than self

  int j,nei;
  int nevent = 0;

  //Nearest-neighbor sampling
  for (j = 0; j < numneigh[i]; j++) {
    nei=neighbor[i][j];
    if (spin[i]==spin[nei])
      continue;

    sites[nevent++]=nei;
  }

  if (nevent == 0) return;

  int iran = (int) (nevent*random->uniform());
  if (iran >= nevent) iran = nevent-1;
```

```
double einitial = site_energy(i), qold[4];
euler2quat(i,qold);

spin[i] = spin[sites[iran]];
phi1[i] = phi1[sites[iran]];
phi2[i] = phi2[sites[iran]];
Phi[i]  = Phi[sites[iran]];

double efinal = site_energy(i), qnew[4];

//Determing misorientation between ij states to
//calculate mobility
double thetar;
int smin = MIN(oldstate,spin[i]);
int smax = MAX(oldstate,spin[i]);

std::pair <int, int> spins = std::make_pair(smin,smax);

// ratio of theta/theta_m
if (misos.count(spins) == 1)
  thetar=misos[spins];
else {
  euler2quat(i,qnew);
  thetar=quaternions(qold,qnew)/thetam;
  misos[spins]=thetar;
}

double p0=(1.0-exp(-bmob*pow(thetar,nmob)));

//Check for isotropic case
if (thetam<1e-8) p0=1;

// accept or reject via Boltzmann criterion
if (efinal <= einitial) {
```

```
      if ((thetar < 1e-8) || (random->uniform() < p0)) {
      }
      else {
        spin[i] = oldstate;
        phi1[i] = iphi[0];
        phi2[i] = iphi[2];
        Phi[i]  = iphi[1];
      }
    }
    else if (temperature == 0.0) {
      spin[i] = oldstate;
      phi1[i] = iphi[0];
      phi2[i] = iphi[2];
      Phi[i]  = iphi[1];
    }
    else if (random->uniform() > p0*exp((einitial-efinal)*t_inverse)) {
      spin[i] = oldstate;
      phi1[i] = iphi[0];
      phi2[i] = iphi[2];
      Phi[i]  = iphi[1];
    }

    if (spin[i] != oldstate) naccept++;
  }
```

Appendix B. AppPottsRS Class Header File

```
/* -------------------------------------------------------
 AppPottsRS class header

 Developed by Efrain Hernandez-Rivera (2017--2018)
 US Army Research Laboratory
 --
 THIS SOFTWARE IS MADE AVAILABLE ON AN "AS IS" BASIS
 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, NEITHER
 EXPRESSED OR IMPLIED
------------------------------------------------------- */

#ifdef APP_CLASS
AppStyle(potts/rs,AppPottsRS)

#else

#ifndef SPK_APP_POTTS_RS_H
#define SPK_APP_POTTS_RS_H

#include <map>
#include "app_potts.h"

namespace SPPARKS_NS {

class AppPottsRS : public AppPotts {
 public:

  AppPottsRS(class SPPARKS *, int, char **);
  ~AppPottsRS();

  void init_app();
  void grow_app();
  void input_app(char *, int, char **);

  virtual void site_event_rejection(int, class RandomPark *);
```

Approved for public release; distribution is unlimited.

```
  double site_energy(int);

 protected:
  double *phi1,*phi2,*Phi; //pointer to 3 rotation angles
  int *spin;
  double thetam;           //High-low angle divider
  double Jij;              //Interaction energy
  int Osym;                //Symmetry Operator flag
  double **symquat;        //Symmetry Operator in quaternion space

  //Mobility = Mm * [1 - exp(-B * {theta/theham}^n)]
  double nmob;            //Mobility exponential power, n
  double bmob;            //Mobility exponential scaling, B

  //Get misorientation angle from quaternions
  double quaternions(const double qi[4], const double qj[4]);

  //Multiplication between quaternion vectors
  void quat_mult(const double qi[4], const double qj[4], double q[4]);

  //Define the symmetry operator based on symmetry flag
  void symmat(double ***);

  //Convert symmetry operator into quaternion space
  void mat2quat(const double O[3][3], double q[4]);
  void euler2quat(int i, double q[4]);

  //map to store misorientations
  std::map<std::pair<int,int>, double> misos;
};

}

#endif
```

```
#endif

/* ERROR/WARNING messages:

E: Illegal ... command

Self-explanatory.  Check the input script syntax and compare to the
documentation for the command.  You can use -echo screen as a
command-line option when running SPPARKS to see the offending
line.

E: One or more sites have invalid values

The application only allows sites to be initialized with specific
values.

*/
```

List of Symbols, Abbreviations, and Acronyms

MCS	Monte Carlo step
PMC	Potts Monte Carlo
SPPARKS	Stochastic Parallel PARticle Kinetic Simulator

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
2 (PDF)	DIR ARL IMAL HAR RECORDS MGMT RDRL DCL TECH LIB
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (PDF)	DIR USARL RDRL WMM B E HERNANDEZ